

UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA
Dipartimento di Scienze Fisiche, Informatiche e Matematiche

Corso di Laurea in Informatica, Big Data Analytics

Relazione attività Neo4j

Studente:

Lorenzo Stigliano
Matricola 185534

Anno Accademico 2022-2023

Contents

| | | |
|----------|--|----------|
| 1 | Introduzione | 3 |
| 1.0.1 | Struttura della relazione | 6 |
| 2 | OpenStreet Map + Neo4j | 7 |
| 2.1 | Implementazione Tecnica | 10 |
| 2.2 | Operazioni di modifica | 12 |
| 2.2.1 | Aggiunta/Modifica nodi | 12 |
| 2.2.2 | Creazioni relazioni | 14 |
| 2.2.3 | Creazione Indici | 14 |
| 2.3 | Interrogazioni | 16 |
| 2.3.1 | Ricerca di un punto di ristoro | 16 |
| 2.3.2 | Ricerca del cinema più vicino ai punti di ristoro | 16 |
| 2.3.3 | Ricerca del campo da baseball più vicino nell'arco di 400 metri da <i>Delacorte Theater</i> | 16 |
| 2.3.4 | Distanza tra <i>Le Pain Quotidien</i> e <i>Zoo School</i> | 17 |
| 2.3.5 | Numero di statue presenti all'interno del parco dal 1880 al 2000, escludendo dal 1971 al 1979 | 17 |

1 Introduzione

Il tema della relazione concerne il progetto OpenStreetMap inserito in un contesto applicativo come Neo4j. OpenStreetMap è un progetto di informazione geografica volontaria, nato nel 2004 in Gran Bretagna dallo studente di dottorato Steve Coast che non riusciva ad ottenere dati geografici open source per i suoi studi e ha iniziato una raccolta diretta e con la conseguente creazione di un database.



Figure 1: Steve Coast

L'idea di OpenStreetMap è simile a Wikipedia, ma in formato geografico. Il progetto è aperto a tutti gli utenti e si pone come database globale. OpenStreetMaps non è solo una alternativa a Google Maps, per quanto assolve a compiti comuni, ma si possono fare molte attività aggiuntive, prima fra tutti: contribuire attivamente alla mappatura. Non esiste una mappatura uniforme di tutte le aree del mondo. Sicuramente le aree più industrializzate e con maggiore presenza umana, godono di una maggiore attenzione, ma ci sono zone del mondo che spesso, forte del poco interesse economico, non sono adeguatamente mappate, per quanto pullulino di vita. Prendiamo il caso delle baraccopoli in Nigeria, che spesso vengono poco curate da parte di Google, ma che pullulano di persone e attività commerciali.

Il progetto si basa sul contributo degli utenti, in maniera *bottom up*, che vivono i luoghi che vengono mappati. Esiste un coordinamento generale, che organizza le varie attività e fa valutazioni in termini di sostenibilità economica del progetto, ma **non controlla gli utenti che partecipano al progetto**. Non sono gestiti quindi i progetti software e decise in maniera esplicita le aree che vengono mappate. Funziona tutto basandosi su un database aperto (open data) su licenza: Database License (ODbL).

La community è distribuita in maniera molto disomogenea e alla situazione attuale su 8 milioni di utenti registrati circa 1 milione sono contributori effettivi, con in media 50 mila contribuzioni al giorno. A contribuire al progetto non ci sono solamente gli utenti, ma anche molte aziende che utilizzano il servizio per strutturare dei propri prodotti. Primi fra tutti Meta che utilizza anche dati da OpenStreetMap per le mappe che vengono visualizzate all'interno del proprio social network Facebook e anche Amazon, che ha una pagina del wiki di OpenStreetMap in raccoglie dati su utenti che mappano le zone limitrofe alle proprie abitazioni per poter effettuare le consegne in maniera migliore nelle aree che mancano di una mappatura efficace.



OpenStreetMap

Figure 2: OpenStreetMap Logo

Per sviluppare questa tesi verrà utilizzata una istanza di questo progetto: l'area di Central Park a New York, resa disponibile per poter essere utilizzata all'interno di Neo4j. Il progetto Neo4j è un database (più precisamente DBMS) sviluppato dall'omonima azienda che ha la particolarità di essere strutturato come un grafo. Un database di questo tipo non ha più colonne e righe, ma possiede nodi con archi che li connettono reciprocamente definendo relazioni. L'esempio classico di un database basato su grafo può essere la rete di amicizie che utente ha costruito all'interno di un social network: immaginiamo che ogni utente sia un nodo e che la relazioni di amicizia che intercorre tra un utente ed un altro siano la coppia di archi che collega in entrambi i versi due nodi. Neo4j è sviluppato in Java, ma è usufruibile anche attraverso linguaggi di programmazione attraverso il sistema proprietario di interrogazione: Cypher + HTTP/Bolt protocol. Molte aziende negli anni hanno iniziato ad adottare questo nuovo tipo di DBMS: Ebay, Levis, Comcast, Adobe, etc. ...



Figure 3: Neo4j Logo

Per poter effettuare le interrogazioni nei confronti di questo DBMS viene utilizzato un linguaggio specifico denominato: Cypher.

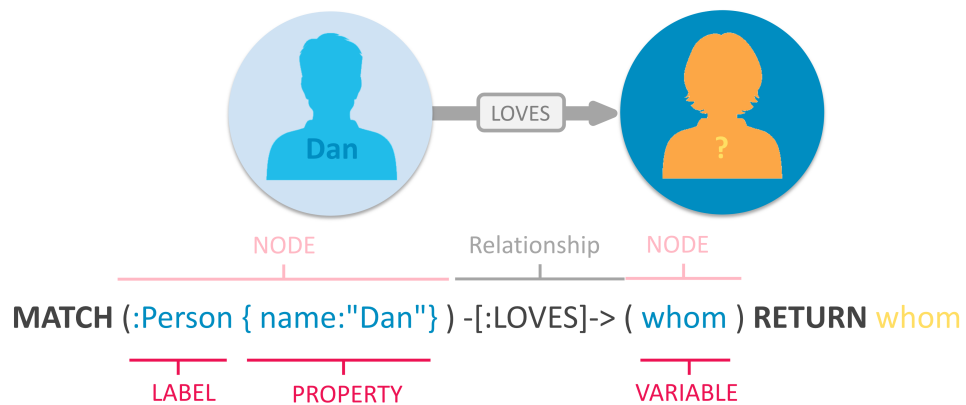


Figure 4: Cypher Example

Questo ci permette di poter esplorare il grafo ed effettuare interrogazioni utilizzando delle parole chiavi come: MATCH, WHERE, WITH, ... per poter ricavare le informazioni necessarie dal database.

1.0.1 Struttura della relazione

La relazione è composta da un due capitoli principali:

1. Introduzione: contiene le informazioni generali riguardo i temi che riguardano la relazione con un cappello introduttivo sul quel che concerne OpenStreetMap e Neo4j
2. OpenStreetMap + Neo4j: mostra il caso d'uso specifico in cui il DBMS Neo4j viene utilizzato su un dataset di OpenStreetMap. Questo è a sua volta composto da due macro sezioni (divise in sotto capitoli)che definiscono le operazioni svolte all'interno del dataset:
 - (a) Operazioni di aggiunta/modifica nodi
 - (b) Interrogazioni

In incipit di questa sezione è presenta una piccola nota tecnica implementativa che spiega come sono stati effettuati i test sul database.

2 OpenStreet Map + Neo4j

Come definito all'interno della introduzione verranno effettuato varie operazioni di esplorazione e manipolazione all'interno di un dataset che rappresenta una porzione di mappa corrispondente al più grande parco pubblico nel distretto di Manhattan, a New York: Central Park. Questo si trova nella Uptown al centro tra i due quartieri residenziali, l'Upper West Side e l'Upper East Side, i quali prendono il nome dalla loro posizione rispetto al parco. Anche le strade che lo circondano prendono il suo nome. È uno dei parchi cittadini più conosciuti del mondo, grazie anche alle sue comparse in numerosi film e telefilm. È di forma rettangolare, lungo circa 4 chilometri e largo 800 metri, ed è conosciuto come il "polmone verde" di New York.



Figure 5: Central Park

A questo punto si suppone di voler utilizzare Neo4j per poter ottenere delle informazioni utili all'esplorazione del parco e l'identificazione di punti di interesse utili. Il parco viene quindi trasformato in un grafo in cui abbiamo vari tipi di nodi:

- OSMNode: nodo generico
- PointOfInterest: nodo che rappresenta un punto di interesse
- Intersection: nodo che aggiunge informazioni geografiche ad un punto di interesse
- OSMTags: nodo informativo rispetto ad un punto di interesse

Solitamente un nodo relativo ad un singolo punto di interesse è connesso a dei nodi gemelli che ne ampliano il contenuto informativo attraverso diversi tipi di archi come:

- ROUTE: definisce la connessione tra un punto di interesse ed un altro
- TAGS: specifica il legame tra un nodo OSMTags e il suo corrispettivo PointOfInterest

Tipicamente ciascuno di questi nodi ha delle informazioni che si basano sulla sua posizione geografica:

- lat: latitudine;
- lon: longitudine;
- location: latitudine + longitudine in relazione alle x e le y di un ipotetico piano cartesiano;

E dei dati, che possono essere contenuti anche all'interno del rispettivo OSM-Tags, che forniscono informazioni di carattere generale come:

- name: nome dell'autore dell'installazione (in caso di statue o opere d'arte);
- year: anno di inserimento all'interno del parco o di creazione;
- type: tipologia del punto di interesse;
- website: sito di riferimento per visualizzare il punto online;
- tourism: definisce la tipologia di turismo di riferimento (per le opere d'arte o installazioni varie assume il valore di artwork);

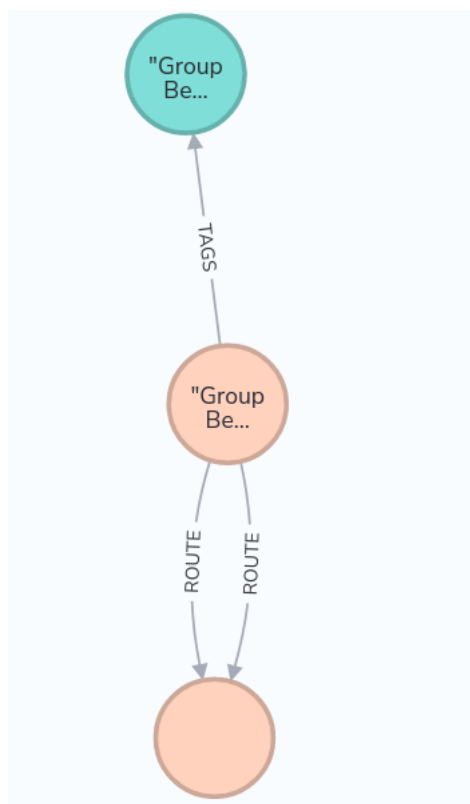


Figure 6: Il nodo più in basso fornisce informazione di tipo geografico, quello più in alto è un OSMTags e quello centrale rappresenta il punto di interesse

Il tutto compone un grafo che conta 189 PointOfInterest, 29641 OSMNode, 3617 OSMTags, 2502 Intersection. Ci sono anche altri nodi che definiscono alcuni passaggi geografici all'interno del parco, ma per un maggior approfondimento si consiglia di leggere il capitolo seguente e effettuare test con mano riguardo il database.

2.1 Implementazione Tecnica


Per effettuare i test è stato utilizzato il dataset presente al seguente link: <https://github.com/neo4j-graph-examples/openstreetmap>, questo corrisponde alla demo presente in formato sandbox sul sito <https://neo4j.com/sandbox/>. Si consiglia di scaricare la repository ed esportare il file: `/openstreetmap/data/openstreetmap-43.dump` all'interno del portale Neo4j AuraDB (<https://neo4j.com/cloud/platform/aura-graph-database/>).



Neo4j's fully managed cloud database service

Figure 7: Neo4j AuraDB

Questo sarebbe una versione online di Neo4j Desktop che ci permette di usufruire gratuitamente di una istanza del nostro database sul cloud, con un dataset grande al massimo 4gb. Si è scelto di sfruttare questo sistema poiché Neo4j for Desktop risulta avere delle pessime performance (soprattutto per le prime minor release della versione 4; migliora sensibilmente dall 5.0 in poi). AuraDB fornisce inoltre anche dei template per poter andare ad implementare il nostro database attraverso vari linguaggi di programmazione: python, java, golang, ...


auraDB

AuraDB
AuraDS
GET HELP

Connect
Snapshots
Import Database
Logs

Python

JavaScript

GraphQL

Java

Spring Boot


.NET

Go

```

1 from neo4j import GraphDatabase
2 import logging
3 from neo4j.exceptions import ServiceUnavailable
4
5 class App:
6
7     def __init__(self, uri, user, password):
8         self.driver = GraphDatabase.driver(uri, auth=
9
10    def close(self):
11        # Don't forget to close the driver connection
12        self.driver.close()
13
14    def create_friendship(self, person1_name, person2_name):
15        with self.driver.session(database="neo4j") as session:
16            # Write transactions allow the driver to
17            result = session.execute_write(
18                self._create_and_return_friendship,
19                person1_name, person2_name)
20            print(f"Created friendship between: {person1_name}, {person2_name}")

```



Prerequisites

- Python 3.7+

Downloading and Installing the Neo4j Driver

Install the Neo4j Driver using pip:

pip install neo4j

Running the example

Figure 8: Neo4j AuraDB Implementazione in Python

2.2 Operazioni di modifica

2.2.1 Aggiunta/Modifica nodi

Si immagini di aggiungere un nodo al grafo; questo rappresenta una installazione di tipo statua, localizzata all'interno del parco e poi di volere visualizzare che sia stata correttamente aggiunta:

```
1 CREATE (n:PointOfInterest {lat: '43',  
2 location: 'point({srid:4327, x:-75, y:43})', lon: '-75', name: 'FIM',  
3 type: 'statue'})
```

Come tutti i nodi PointOfInterest è composto da una ulteriore label:

- OSMNode

E con la seguente istruzione, sfruttando il comando *SET* è possibile aggiungere tale informazione al nodo:

```
1 MATCH (n:PointOfInterest)  
2 WHERE n.name = 'FIM'  
3 SET n:OSMNode
```

Come detto nel capitolo precedente risulta necessario creare anche un altro nodo che contenga indicazioni di tipo geografica (con l'aggiunta di una ulteriore label: *Intersection*) e un nodo OSMTags:

```
1 CREATE (n:OSMNode {lat: '43', location: 'point({srid:4327, x:-75, y:43})',  
2 lon: '-75'})  
3  
4 MATCH (n:OSMNode)  
5 WHERE n.lat = '43' AND n.lon = '-75'  
6 SET n:Intersection  
7  
8 CREATE (n:OSMTags {artist_name: "Dipartimento FIM",  
9 artwork_type: "statue", name: "FIM", start_date: "2008",  
10 tourism: "artwork"})
```

Il risultato se si visualizzasse il risultato di queste operazioni questo sarebbe il risultato:

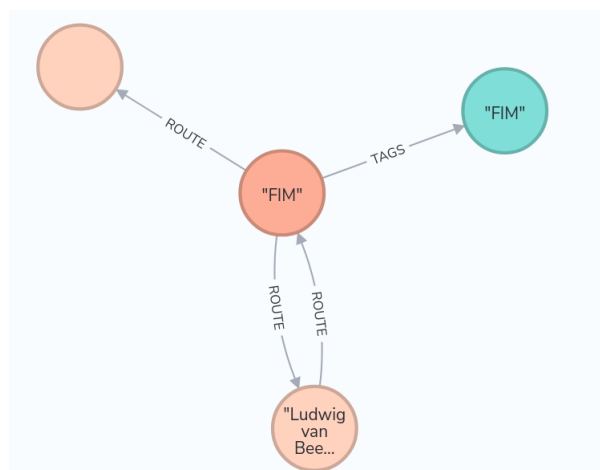


Figure 9: FIM Node

Nella prossima sezione vedremo come andare a creare delle relazioni e realizzare parte dei collegamenti che interessano il nodo FIM in questa immagine.

2.2.2 Creazioni relazioni

Si consideri di creare una relazione tra l'installazione denominata FIM e quella dedicata a Ludwig van Beethoven. Questa è composta da archi in ambo i versi per le due installazioni:

```
1 MATCH
2   (a:PointOfInterest),
3   (b:PointOfInterest)
4 WHERE a.name = 'FIM' AND b.name = 'Ludwig van Beethoven'
5 CREATE (b)-[r:ROUTE]->(a)
6 RETURN type(r)
7
8 MATCH
9   (a:PointOfInterest),
10  (b:PointOfInterest)
11 WHERE a.name = 'FIM' AND b.name = 'Ludwig van Beethoven'
12 CREATE (a)-[r:ROUTE]->(b)
13 RETURN type(r)
```

Si immagini di volere collegare il nodo OSMTags e OSMNode (Intersection) creato nella sezione precedente al corrispettivo PointOfInterest:

```
1 MATCH
2   (a:PointOfInterest),
3   (b:OSMTags)
4 WHERE a.name = 'FIM' AND b.name = 'FIM'
5 CREATE (a)-[r:TAGS]->(b)
6 RETURN type(r)
7
8 MATCH
9   (a:PointOfInterest),
10  (b:OSMNode)
11 WHERE a.name = 'FIM' AND b.lon = '-75' AND b.lat = '43'
12 CREATE (a)-[r:ROUTE]->(b)
13 RETURN type(r)
```

2.2.3 Creazione Indici

Considerando che molte delle interrogazioni si basano sul fatto che un utente possa andare ad effettuare una ricerca basata sul nome di alcuni luoghi significativi del parco è consigliabile creare un indice che sui nomi dei nodi dei punti di interesse:

```
1 CREATE INDEX FOR (p:PointOfInterest) ON (p.name)
```

Questo ci permette di ottenere performance migliori, che possiamo andare a misurare con una istruzione chiamata PROFILE, che è possibile eseguire prima di una interrogazione:

```
1 PROFILE
2 MATCH (p:PointOfInterest {type:'clock'})
3 RETURN p.name
```


2.3 Interrogazioni

2.3.1 Ricerca di un punto di ristoro

Si consideri che un utente voglia valutare quali punti di ristoro sono disponibili all'interno dell'area di Central Park: la ricerca si basa sul fatto che si stanno esplorando tutti i nodi `PointOfInterest` in cui il campo `type`, che indica la tipologia del punto di interesse, sia *cafe*:

```
1 MATCH (p:PointOfInterest {type:'cafe'})
2 RETURN p.name
```

Se volessi considerare solamente il numero di punti ristoro è sufficiente utilizzare la funzione `count` nell'atto del `RETURN`:

```
1 MATCH (p:PointOfInterest {type:'cafe'})
2 RETURN count(p)
```

2.3.2 Ricerca del cinema più vicino ai punti di ristoro

Si consideri di voler cercare il cinema che è più vicino ad un punto ristoro qualunque:

```
1 MATCH path = (p1:PointOfInterest {type:'cafe'}),
2 (p2:PointOfInterest {type:'theater'})
3 CALL apoc.algo.dijkstra(p1, p2, 'ROUTE', 'distance') YIELD weight AS dist
4 RETURN p1.name, p2.name, dist ORDER BY dist
```

In questo caso si sta usando APOC. Questa è una libreria aggiuntiva per Neo4j che fornisce centinaia di metodi e funzioni utili che espandono le potenzialità di Neo4j. In questo caso specifico si sta utilizzando un metodo che offre la possibilità di andare a fare una esplorazione anche in termini di distanza geografica reale sui nodi.

2.3.3 Ricerca del campo da baseball più vicino nell'arco di 400 metri da *Delacorte Theater*

Si immagini di cercare il campo da baseball più vicino al palco del *Delacorte Theater*, nell'arco di 400 metri:

```
1 MATCH path = (p1:PointOfInterest {type:'baseball'}),
2 (p2:PointOfInterest {name: 'Delacorte Theater'})
3 CALL apoc.algo.dijkstra(p1, p2, 'ROUTE', 'distance') YIELD weight AS dist
4 WHERE dist < 400
5 RETURN p1.name, p2.name, dist ORDER BY dist
```

2.3.4 Distanza tra *Le Pain Quotidien* e *Zoo School*

Si consideri l'eventualità di cercare il punto lo zoo più vicino partendo dal punto di ristoro: *Le Pain Quotidien*.

```
1 MATCH path=shortestpath((p1:PointOfInterest {name:'Le Pain Quotidien'})
2 -[:ROUTE*]-(p2:PointOfInterest {name:'Zoo School'}))
3 WITH relationships(path) AS rels
4 UNWIND rels AS rel
5 RETURN sum(rel.distance)
```

In questo caso abbiamo utilizzato la funzione *shortestpath* proviamo a verificare se la distanza ottenuta: 2287.9036630448454, coincide utilizzando *apoc.algo.dijkstra*.

```
1 MATCH path = (p1:PointOfInterest {name:'Le Pain Quotidien'}),
2 (p2:PointOfInterest {name:'Zoo School'})
3 CALL apoc.algo.dijkstra(p1, p2, 'ROUTE', 'distance') YIELD weight AS dist
4 RETURN p1.name, p2.name, dist ORDER BY dist LIMIT 1
```

Utilizzando la seguente interrogazione si riscontra come la distanza sia: 1040.0073911153404. Questa discrepanza è dovuta al fatto che la prima funzione calcola il percorso migliore in base al minor numero di archi attraversati. Questo non sempre coincide realmente con una minore distanza geografica. Nel primo caso è come se valutassimo un algoritmo per grafi non pesati all'interno di un grafo pesato. Il risultato che si deve considerare come vero è quello ottenuto con Dijkstra perché considera il peso degli archi e non il numero di archi attraversati.

2.3.5 Numero di statue presenti all'interno del parco dal 1880 al 2000, escludendo dal 1971 al 1979

Si consideri la possibilità di indagare riguardo il numero di statue risalenti al periodo che intercorre dal 1880 al 2000, escludendo dal 1971 al 1979:

```
1 MATCH (t:OSMTags)
2 WHERE t.start_date >= "1880" AND t.start_date <= "1970"
3 AND t.artwork_type = "statue"
4 RETURN count(t)
5 UNION
6 MATCH (t:OSMTags)
7 WHERE t.start_date >= "1980"
8 AND t.start_date <= "2000" AND t.artwork_type = "statue"
9 RETURN count(t)
```

In questa interrogazione viene utilizzato il nodo *OSMTags* perché contiene informazioni rispetto alla data in cui l'installazione è stata costruita.

References

- [1] Manuale Neo4j V5, <https://neo4j.com/docs/cypher-manual/current/>
- [2] Materiale OpenStreetMap, <https://github.com/neo4j-graph-examples/openstreetmap>
- [3] Loading OpenStreetMap data into a Graph Database (Taylor Callsen), <https://taylor.callsen.me/loading-openstreetmap-data-into-a-graph-database/>